Orion File System : File-level Host-based Virtualization

Amruta Joshi
Pune Institute of Computer Technology,
Dhankavadi, Pune 411043, India
020-2437-1101
amruta_pict@yahoo.com

Faraz Shaikh
Pune Institute of Computer Technology,
Dhankavadi, Pune 411043, India
020-2437-1101
faraz_irulz@yahoo.co.in

Sapna Todwal
Pune Institute of Computer Technology,
Dhankavadi, Pune 411043, India
020-2437-1101
sapnatodwal@yahoo.co.in

Abstract— The aim of Orion is to implement a solution that provides file-level host-based virtualization that provides for better aggregation of content/information based on semantics and properties. File-system organization today very closely mirrors storage paradigms rather than user-access paradigms and semantic grouping. All file-system hierarchies are containers that are expressed based on their physical presence (a separate drive letter on Windows, or a particular mount point based on the volume in Unix).

We have implemented a solution that will allow users to organize their files based on their convenience. We define this convenience in the following forms:
* The ability to organize the namespace based on certain attribute properties (file-system metadata virtualization)
* The ability to de-link position of a file in the hierarchy from its actual storage (file metadata virtualization)
* The ability to create and manipulate namespaces using well-known metaphors (XML schema descriptions and schema editors)
* The ability to continue using the standard metaphors for manipulation and access to information (file-system kernel API's), thus maintaining current large body of applications unbroken)

Currently, no other solution allows users to organize their files using convenient semantic groupings while continuing to use standard applications. This solution is unique in the sense that it allows flexible namespace construction using XML.

1. Introduction

ORION is a semantic file system capable of providing file level aggregation according to the semantics of the file and provide for the directory structure virtualization of an existing file system. A semantic file system is an information storage system that provides a flexible associative access to the system's contents by automatically extracting attributes from file with the help of type specific transducers.[5]

Associative access is provided by extension to existing tree-structured file system protocols, and by protocols that are designed specifically for content-based access. Compatibility with existing file system protocols is provided by introducing the concept of a virtual directory. Virtual directory names are interpreted as queries, and thus provide a flexible associative access to files and directories in a manner compatible with existing software. Rapid attribute-based access to system contents is implemented by automatic extraction and indexing of key properties of files. The automatic indexing of files and directories is called "semantic" because user programmable transducers use information about these semantics of files to extract the properties for indexing. The extracted properties are then stored in a relational database so that queries can be run against them. Experimental results from our semantic file system implementation ORION show that semantic file systems present a more effective storage abstraction than the traditional tree structured file systems for information sharing, storage and retrieval.

2. Motivation

When you have a large number of items, it is important to have a flexible and efficient mechanism to search for particular items based on their properties and content.[4] This is exactly what Orion aims to achieve.

Storage Paradigm to User Access Paradigm

Orion takes a file system from storage paradigm to user access paradigm and semantic grouping.[2] Thus Orion takes an FS closer to the way people think about files. We do not think of files according to where it is stored. Rather we observe that people think about files based on their content or properties. Hence, search for file should be based on these rather than their storage location in the directory hierarchy.

Orion uses grouping instead of ordering to locate files. A grouping is interpreted by performing a set intersection of those lists for every object named in the grouping.[3]

Platform rather than an application

Orion is a platform and not just an application. Being a platform, all file system kernel API's remain unchanged. Also existing applications automatically start working with the new platform and can exploit its features without any changes to the applications themselves. Also, Orion is not targeted at any particular file type and can support any file types that may come up in the future.

Multiple Views of the same set of files

Orion aggregates information based on its semantics. It restructures the logical layout of the file system and gives you the ability to have different logical views of the same set of physical files. All files are stored physically in the central object store. Atop this, we have a define and mount multiple views, based on property of the file like its type or some extended attribute of the file like its owner, movie of an mp3, etc.

3. Approach

We have developed an approach for information storage that both permits users to share information more effectively, and provides reductions in programming effort and program complexity. To be effective, this new approach needs a transition path from existing file systems. To achieve this goal we will develop a file system, which will provide for file aggregation according to semantics and virtualization of the file system's directory structure.

Associative access is designed to make it easier for users to share information by helping them discover and locate programs, documents, and other relevant objects. For example, files can be located based upon transducer-generated attributes such as author, exported or imported procedures, words contained, type and title. ORION as a semantic file system implementation is totally transparent to the legacy applications which access the files on our file system via the normal Linux file system API's. Another approach can be making a complete user level program, which provides a new interface altogether. Though this approach is the easiest, it would require all new programs to adhere to the newly introduced interface and nevertheless all the existing legacy applications would fail.

ORION thus integrates associative access into a tree structured file system through the concept of a virtual directory. Virtual directory names are interpreted as queries and thus provide flexible associative access to files and directories in a manner compatible with existing software. For indexing the semantic attributes of the files ORION uses a relational database with an extensible schema. So the search power of a relational engine is used when specifying the queries to be associated the virtual directories. The project thus merges the advantages from the two most prominent storage technologies the file system and the database to give the user a solution in the form a file system with database like extension.[1]

The bottom line is ORION is an implementation prototype for further semantic file systems. ORION's interface is strictly backward compatible to traditional tree structured file system as we understand the "people don't like abrupt paradigm changes" specially when it comes to storing their critical data.

4. Architecture

ORION consists of 3 major parts viz. Object Store, Orion View Core and Database Module. The object store and orion view core operate below the Linux VFS layer while the database is in user space. The object store is a flat file system stacked atop a disk file system. Every file created in the object store has to have a corresponding record in the database. Hence an updating thread is activated to log in details of the newly created file. It communicates with the update daemon which actually updates the database. The Orion view core is responsible for creating virtual directories and listing files in them. Every directory has a query associated with it. To find out which files satisfy the query, Orion view core uses the recordset cache. If the entry is not found in this cache, the record is searched for in the database by using the cache fault handler.

ORION provides the user with two different types of attributes viz. Normal attributes and Extended attributes.

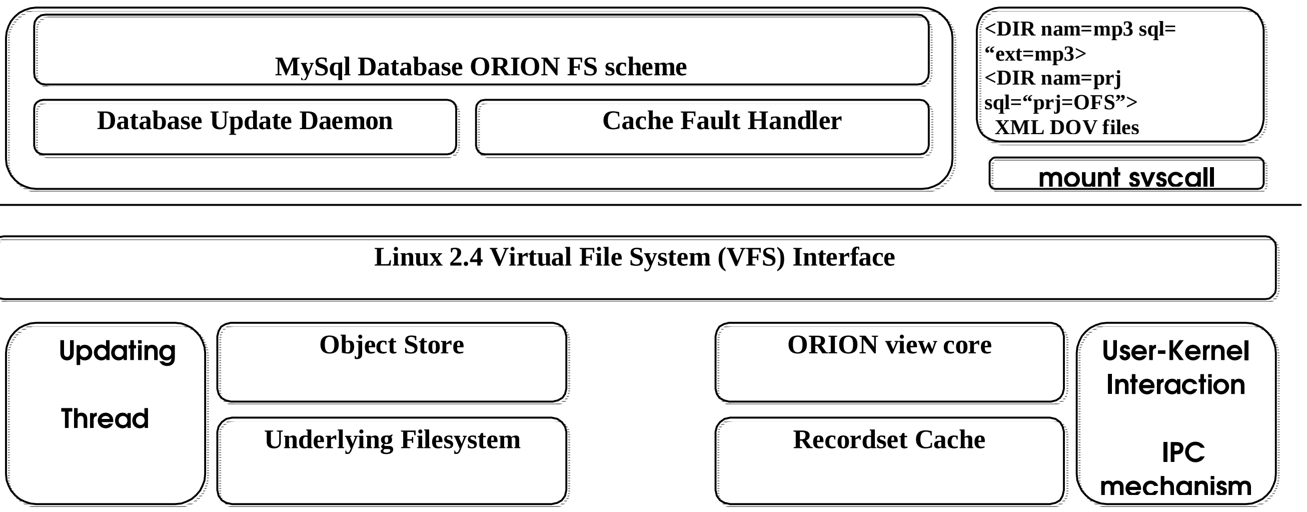

Fig 1: ORION Architecture

The normal attributes are the attributes like filename, owner, size, uid, gid etc which are provided to us by the underlying file system. The extended attributes, on the other hand, are provided by the user in the form of name value pairs. The main power of Orion lies in these extended attributes.

Orion uses an extensible schema for storing of normal as well as extended attributes, catagorised in four major datatypes viz. text, date_time, number and boolean. Orion has five tables in the database. The attribute_record table, containing filename, inode number and file type, is the base table with exactly one entry for each file in the object store. This table has one to many relationship with the four other tables in the database. These child tables store the normal and extended attributes of the files based on their datatype. The child tables contain inode number as the foreign key referring to the inode number in the arrtibute_record table, the attribute name and the attribute value. Thus the attribute names are not made as column names as in case of regular method of storing attributes but extensible schema is used. So the structure is essentially vertical than horizontal.

then the control initially passes down to the VFS layer in the function vfs_getdents(). Then the control trickles down to the filesystem specific readdir i.e ofs_readdir.

In step 1 ofs_readdir() asks for the recordset corresponding to Cfiles in the recordset cache. If there is a cache hit then the pointer to the recordset is returned.
Else as shown in step 2 , in case of cache fault the query for the directory and the pid of the process waiting for listing is sent to the user space deamon called cache fault handler. Corresponding to this request a thread is executed in the

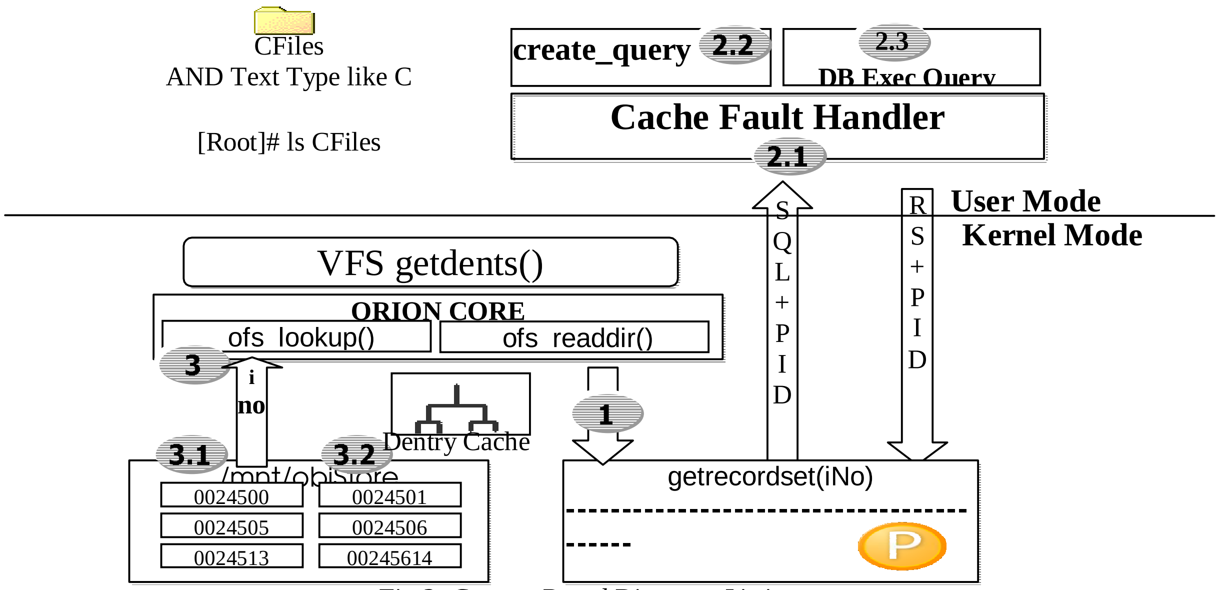

Fig 2: Content Based Directory Listing

5. IMPLEMENTATION

As the initial setup the object store is present on an partition and the meta-data about the files in the object store is stored in the database. After this the content-based access is setup in three phases.

- Creating the virtual directory structure.
- Content based Listing for the virtual directories.
- Logging the activities on the object store in the relational database.

Creating the virtual directories
The information about the directory structure to be mounted on top of the object store is stored in file. This file is in XML format and is called data organization view file or the DOV file. The user the issues the mount(2) system call with two extra options, the path of the object store and the name of the DOV file, to mount the directory structure specified in the DOV file. The virual directory is of the form "directory name and its associated query". For eg: if we need a virtual directory containing all the c files we will name the directory as "Cfiles" and will associate it with the query "And Text type like c".

Content Based Directory Listing (Refer fig 2)
Suppose there is a virtual directory called Cfiles with the associated query "And Text type like c". Now when an application like ls tries to do listing of Cfiles as "ls Cfiles"

deamon in step 2.1. As seen in step 2.2 this query is converted to a SQL query . In step 2.3 this query is executed and the duplicates are resolved.Finally in step 2.4 the recordet cache is populated with the recordset of the form "inode-no. filename" and the process waiting for the recordset is woken up.

Now when ofs_lookup actually does lookup in step 3.1 on the directory Cfiles, the inodes are extracted from the objectstore with the inode numbers in the recordset cache using function iget(sb,inode-no). Then finally in step 3.2 the dentry cache is populated with the names attached using function d_add().

Logging Activities on the Object Store (Refer Fig 3)
Now whenever a new file is created or edited on the views in the Orion file system, the appropriate updates have to be done in the database existing in the user space, i.e the activites happening in the object store have to be logged on into the database.

Lets take an example. Suppose an application like touch creates a new file called newfile.xyz , then the control initially passes down to the function vfs_create() . It then passes the control to the file system specific create function ofs_create(). In step1 the ofs_create() function makes the vnodes i.e the dentry and the inode compatible with the underlying file system. After this when the control actually goes down to the create function of the underlying file system, the call is trapped. In the precall of create (step 2),

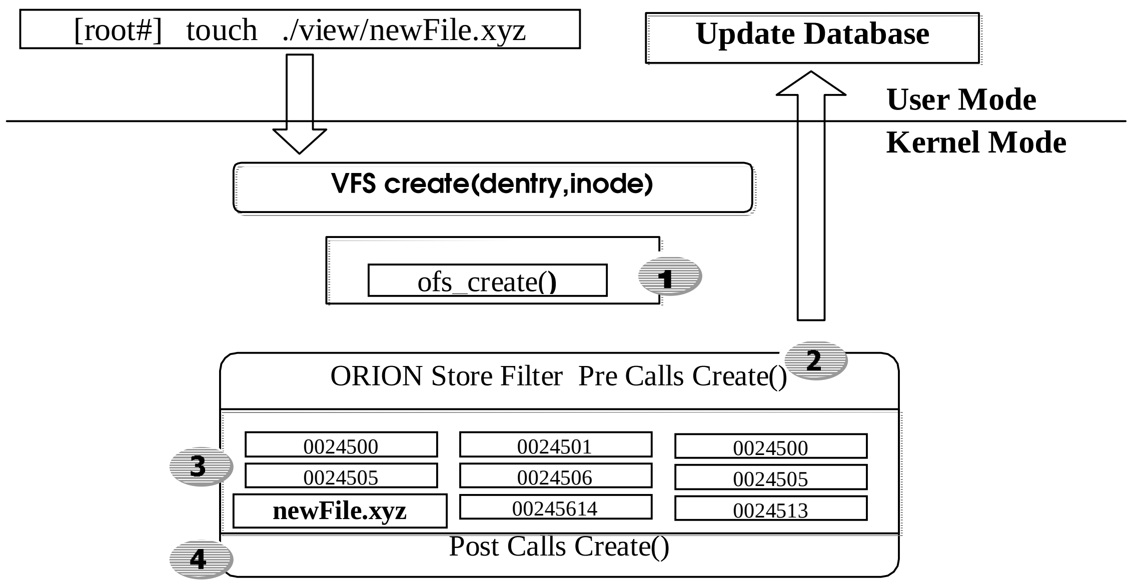

Fig 3: Logging Of Activities on the object store

the Orion store filter updates the database to reflect the changes in the object store. Similarly even other system calls like unlink, set_attr and rename are trapped to make appropriate updates in the database.

Now in this example, in the actual create call in step 3 the file is actually created with its original name i.e newfile.xyz. In the postcall of create i.e in step 4, the name newfile.xyz is renamed by its inode number in the flat object store so that the new file created becomes a part of the object store. Along with this the process (here "touch") which invoked the create operation is woken up which otherwise is sleeping through out the process. Thus the created new file is registered into the database.

Thus Orion is a filesystem empowered with querying abilities on file's extended attributes and virtualization of directory structure.

6. CONCLUSION

Thus after the completion of the "ORION Filesystem" we have successfully implemented a semantic file system. All the requirements given in the scope of the project were met. Initial performance results show that adding a semantic access protocol over a traditional file system doesn't add much overhead. The major gain of using such a file system is organization of data is a cleaner and easier way. With hard disk sizes reaching up to 100 GB and people storing more and more semantic rich data on their file system, a semantic file system capable of organizing its contents automatically is definitely a must have. No wonder Microsoft is spending a lot of money in the research and development of their own semantic Filesystem WINFS.

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude our guide, ***Mr. Anandamoy Roy Chowdhary, Calsoft Inc. Pvt. Ltd, Pune*** for his excellent guidance and direction. We would also like to thank our internal guide, ***Dr. C.V.K. Rao, PICT, Pune*** for the constant encouragement and assistance he provided us at every stage of the project.

REFERENCES

[1] "Practical File System Design with the Be File System", 1st Edition By Dominic Giampaolo, Morgan Kaufmann Publications.
[2] "File System and Storage Advancement in Windows Longhorn" by Q. Clark
[3] "The Naming System Venture", Hans Reiser (Jan 2001)
[4] "Revolutionary File Storage System Lets Users Search and Manage Files Based on Content", Richard Grimes.
[5] "Semantic File Systems", David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, James W. O'Toole, Operating Systems Review, v25 n5 1991